Analýza algoritmů. Složitost.

Matematické algoritmy (11MA)

Jan Přikryl

Ústav aplikované matematiky
ČVUT v Praze, Fakulta dopravní

6. přednáška K611MA
čtvrtek 4. listopadu 2010

verze: 2010-11-03 16:17

Obsah přednášky

Algoritmy a programy
Co je co

Algoritmus:

- myšlenka řešení nějakého problému
- konečný počet kroků řešení
- vyjadřujeme nejčastěji slovně nebo pseudokódem

Program:

- implementace algoritmu ve zvoleném programovacím jazyce

Bez algoritmu nelze napsat program.

Analýza složitosti algoritmů

Proč nás to vlastně zajímá

Ideální kombinace: efektivní algoritmus + efektivní implementace

Analýza algoritmu:

- poskytne **předpověď výkonnosti** algoritmu
- je **vhodnější než experimenty**
- umožní výběr **vhodné varianty** řešení

Asymptotická složitost **paměťová** × složitost **časová**

Analýza složitosti algoritmů (2)

Předpověď chování

Analýza efektivity: Identifikace efektivních a neefektivních částí algoritmu umožní soustředit se na ty části, jejichž úprava přinese nejvyšší nárůst výkonu.

Předpověď výkonnosti programu

Velké projekty potřebují apriorní odhad výkonnosti pro daný hardware – je třeba učinit odhad bez znalosti detailů programového kódu.

Identifikace úzkých míst a jejich vhodné ošetření ještě před vlastním naprogramováním.

Analýza složitosti algoritmů (2)

Předpověď chování

Analýza efektivity: Identifikace efektivních a neefektivních částí algoritmu umožní soustředit se na ty části, jejichž úprava přinese nejvyšší nárůst výkonu.

Předpověď výkonnosti programu

Velké projekty potřebují apriorní odhad výkonnosti pro daný hardware – je třeba učinit odhad bez znalosti detailů programového kódu.
Identifikace úzkých míst a jejich vhodné ošetření ještě před vlastním naprogramováním.

Analýza složitosti algoritmů (3)

Ověření vlastností

Vhodnější než experimenty

Experimenty ověří chování pouze ve vybraných krizových případech – místo „dokázali jsme, že daný algoritmus funguje správně“ lze pouze tvrdit: „nenalezli jsme způsob, jak prokázat, že algoritmus je špatně“.

Záruky funkčnosti poskytuje jedině **formální analýza**.

Výběr vhodné varianty

Ne vždy je nejvhodnější varianta ta, jenž je v počtu instrukcí nejefektivnější a tedy nejrychlejší – *např. implementace pro jednočipový počítač × pracovní stanice.*

Analýza složitosti algoritmů (3)

Ověření vlastností

Vhodnější než experimenty

Experimenty ověří chování pouze ve vybraných krizových případech – místo „dokázali jsme, že daný algoritmus funguje správně“ lze pouze tvrdit: „nenalezli jsme způsob, jak prokázat, že algoritmus je špatně“.

Záruky funkčnosti poskytuje jedině **formální analýza**.

Výběr vhodné varianty

Ne vždy je nejvhodnější varianta ta, jenž je v počtu instrukcí nejefektivnější a tedy nejrychlejší – *např. implementace pro jednočipový počítač* × *pracovní stanice*.

Obsah přednášky

Ilustrační příklad
Fibonacciho posloupnost

Poprvé popsána italským matematikem Leonardem z Pisy, známým také jako Fibonacci (1202).

Růstu populace králíků za poněkud idealizovaných podmínek.

Číslo $F(n)$ popisuje velikost populace po n měsících, předpokládáme-li, že

- první měsíc se narodí jediný pár,
- nově narozené páry jsou produktivní od druhého měsíce svého života,
- každý měsíc zplodí každý produktivní pár jeden další pár,
- králíci nikdy neumírají, nemají predátory.

Ilustrační příklad

Stavy populace králíků

Fibonacciho číslo	Stav
$F(1) = 1$	začínáme s jedním párem
$F(2) = 1$	ještě jsou příliš mladí
$F(3) = 2$	tento měsíc již zplodí první potomky
$F(4) = 3$	druhý pár potomků
$F(5) = 5$	první potomci třetí generace

Obecně

$$F(n) = F(n-1) + F(n-2)$$

Jak efektivně zjistit $F(n)$ pro zvolené n?

Explicitní řešení

Přímé vyjádření $F(n)$

Explicitní nerekurzivní vztah pro n-tý člen Fibonacciho posloupnosti je

$$F(n) = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}},$$

kde ϕ je hodnota zlatého řezu,

$$\phi = \frac{1+\sqrt{5}}{2} = 1{,}61803398874989\cdots \approx 1{,}618.$$

Algoritmus 1

Spočti

$$F(n) = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}.$$

Explicitní řešení

Přímé vyjádření $F(n)$

Explicitní nerekurzivní vztah pro n-tý člen Fibonacciho posloupnosti je

$$F(n) = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}},$$

kde ϕ je hodnota zlatého řezu,

$$\phi = \frac{1+\sqrt{5}}{2} = 1{,}61803398874989\cdots \approx 1{,}618.$$

Algoritmus 1

Spočti

$$F(n) = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}.$$

Analýza Algoritmu 1

Aneb co všechno může dopadnout špatně

Reprezentace čísel v plovoucí řádové čárce má svá omezení. V počítači budeme vztah reprezentovat jako

$$F(n) = \frac{1{,}61803^n - 0{,}61803^n}{2{,}23606}.$$

Jaké budou výsledky?

$F(2) = 1{,}00000$ je v pořádku

$F(3) = 1{,}78884$ zaokrouhlíme na 2

$F(20) = 6764{,}69$ ještě zaokrouhlíme na 6765

$F(21) = 10945{,}4$ už zaokrouhlíme na 10945 místo 10946

$F(25) = 75020{,}6$ by mělo být 75025

Existuje přesnější varianta výpočtu?

Analýza Algoritmu 1

Aneb co všechno může dopadnout špatně

Reprezentace čísel v plovoucí řádové čárce má svá omezení. V počítači budeme vztah reprezentovat jako

$$F(n) = \frac{1{,}61803^n - 0{,}61803^n}{2{,}23606}.$$

Jaké budou výsledky?

$F(2) = 1{,}00000$ je v pořádku
$F(3) = 1{,}78884$ zaokrouhlíme na 2
$F(20) = 6764{,}69$ ještě zaokrouhlíme na 6765
$F(21) = 10945{,}4$ už zaokrouhlíme na 10945 místo 10946
$F(25) = 75020{,}6$ by mělo být 75025

Existuje přesnější varianta výpočtu?

Analýza Algoritmu 1

Aneb co všechno může dopadnout špatně

Reprezentace čísel v plovoucí řádové čárce má svá omezení. V počítači budeme vztah reprezentovat jako

$$F(n) = \frac{1{,}61803^n - 0{,}61803^n}{2{,}23606}.$$

Jaké budou výsledky?

$F(2) = 1{,}00000$ je v pořádku
$F(3) = 1{,}78884$ zaokrouhlíme na 2
$F(20) = 6764{,}69$ ještě zaokrouhlíme na 6765
$F(21) = 10945{,}4$ už zaokrouhlíme na 10945 místo 10946
$F(25) = 75020{,}6$ by mělo být 75025

Existuje přesnější varianta výpočtu?

Analýza Algoritmu 1

Aneb co všechno může dopadnout špatně

Reprezentace čísel v plovoucí řádové čárce má svá omezení. V počítači budeme vztah reprezentovat jako

$$F(n) = \frac{1{,}61803^n - 0{,}61803^n}{2{,}23606}.$$

Jaké budou výsledky?

$F(2) = 1{,}00000$ je v pořádku
$F(3) = 1{,}78884$ zaokrouhlíme na 2
$F(20) = 6764{,}69$ ještě zaokrouhlíme na 6765
$F(21) = 10945{,}4$ už zaokrouhlíme na 10945 místo 10946
$F(25) = 75020{,}6$ by mělo být 75025

Existuje přesnější varianta výpočtu?

Analýza Algoritmu 1

Aneb co všechno může dopadnout špatně

Reprezentace čísel v plovoucí řádové čárce má svá omezení. V počítači budeme vztah reprezentovat jako

$$F(n) = \frac{1{,}61803^n - 0{,}61803^n}{2{,}23606}.$$

Jaké budou výsledky?

$F(2) = 1{,}00000$ je v pořádku
$F(3) = 1{,}78884$ zaokrouhlíme na 2
$F(20) = 6764{,}69$ ještě zaokrouhlíme na 6765
$F(21) = 10945{,}4$ už zaokrouhlíme na 10945 místo 10946
$F(25) = 75020{,}6$ by mělo být 75025

Existuje přesnější varianta výpočtu?

Rekurzivní algoritmus

Výpočet podle definice

Hodnoty $F(0)$ až $F(2)$ předpočítáme, zbytek lze s jejich pomocí vyjádřit.

Algoritmus 2

Require: $n \geq 0$
Ensure: $y = F(n)$
1: **if** $n = 0$ **then**
2: $\quad y \Leftarrow 0$
3: **else if** $n \leq 2$ **then**
4: $\quad y \Leftarrow 1$
5: **else**
6: $\quad y \Leftarrow F(n-1) + F(n-2)$
7: **end if**

Jak efektivní je daný algoritmus?

Rekurzivní algoritmus

Výpočet podle definice

Hodnoty $F(0)$ až $F(2)$ předpočítáme, zbytek lze s jejich pomocí vyjádřit.

Algoritmus 2

Require: $n \geq 0$
Ensure: $y = F(n)$
1: **if** $n = 0$ **then**
2: $\quad y \Leftarrow 0$
3: **else if** $n \leq 2$ **then**
4: $\quad y \Leftarrow 1$
5: **else**
6: $\quad y \Leftarrow F(n-1) + F(n-2)$
7: **end if**

Jak efektivní je daný algoritmus?

Rekurzivní algoritmus
Výpočet podle definice

Hodnoty $F(0)$ až $F(2)$ předpočítáme, zbytek lze s jejich pomocí vyjádřit.

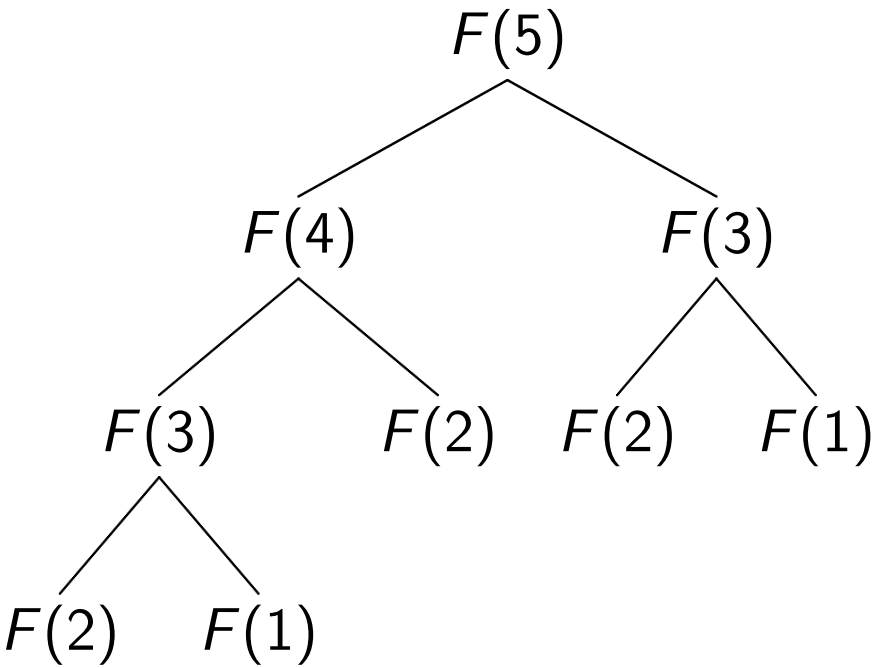
Algoritmus 2

Require: $n \geq 0$
Ensure: $y = F(n)$
1: **if** $n = 0$ **then**
2: $\quad y \Leftarrow 0$
3: **else if** $n \leq 2$ **then**
4: $\quad y \Leftarrow 1$
5: **else**
6: $\quad y \Leftarrow F(n-1) + F(n-2)$
7: **end if**

Jak efektivní je daný algoritmus?

Rekurzivní algoritmus
Efektivita

Posloupnost výpočtu $F(5)$:



Počet kroků pro $F(n)$ je $\tau(n) = 3 \cdot F(n) - 2$.

Dynamické programování
Varianta „shora dolů“

Technika matematické optimalizace.

Dekompozice problému na identické podproblémy.

Dva základní přístupy:

- **shora dolů** – řešíme podproblémy postupně a pamatujeme si řešení
- **zdola nahoru** – vyřešíme všechny potřebné podproblémy a skládáme je

Algoritmus 3

Require: $n \geq 0$
Ensure: $y = F(n)$
1: Alokuj $f[1 \ldots n]$
2: $f[0] \Leftarrow 0$
3: $f[2] \Leftarrow f[1] \Leftarrow 1$
4: **for** $i = 3$ to n **do**
5: $\quad f[i] \Leftarrow f[i-1] + f[i-2]$
6: **end for**
7: $y \Leftarrow f[n]$

Dynamické programování
Varianta „shora dolů“

Technika matematické optimalizace.

Dekompozice problému na identické podproblémy.

Dva základní přístupy:

- **shora dolů** – řešíme podproblémy postupně a pamatujeme si řešení
- **zdola nahoru** – vyřešíme všechny potřebné podproblémy a skládáme je

Algoritmus 3

Require: $n \geq 0$
Ensure: $y = F(n)$
1: Alokuj $f[1 \ldots n]$
2: $f[0] \Leftarrow 0$
3: $f[2] \Leftarrow f[1] \Leftarrow 1$
4: **for** $i = 3$ to n **do**
5: $\quad f[i] \Leftarrow f[i-1] + f[i-2]$
6: **end for**
7: $y \Leftarrow f[n]$

Dynamické programování
Varianta „zdola nahoru“

Algoritmus 3 potřebuje pole n prvků pro uchování minulých členů posloupnosti.
Jde to ale i bez něj.

Algoritmus 4

Require: $n \geq 0$
Ensure: $y = F(n)$

```
1:  if n = 0 then
2:      y ⇐ 0
3:  else
4:      a ⇐ 1, b ⇐ 1
5:      for i = 3 to n do
6:          c ⇐ a + b
7:          a ⇐ b, b ⇐ c
8:      end for
9:      y ⇐ b
10: end if
```

Dynamické programování
Varianta „zdola nahoru“

Algoritmus 3 potřebuje pole n prvků pro uchování minulých členů posloupnosti.
Jde to ale i bez něj.

Algoritmus 4

Require: $n \geq 0$
Ensure: $y = F(n)$

```
1:  if n = 0 then
2:      y ⇐ 0
3:  else
4:      a ⇐ 1, b ⇐ 1
5:      for i = 3 to n do
6:          c ⇐ a + b
7:          a ⇐ b, b ⇐ c
8:      end for
9:      y ⇐ b
10: end if
```

Maticová varianta

Jiná explicitní forma

Pro členy Fibonacciho posloupnosti platí také

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

Použitím opakovaného mocnění snížíme počet kroků na $O(\log n)$.

Maticová varianta

Pomocí opakovaného mocnění

Algoritmus 5

Require: $n \geq 0$
Ensure: $y = F(n)$

1: **if** $n = 0$ **then**
2: $\quad y \Leftarrow 0$
3: **else**
4: $\quad \mathbf{M} \Leftarrow \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$
5: $\quad \mathbf{M} \Leftarrow \text{matpow}(\mathbf{M}, n-1)$
6: $\quad y \Leftarrow M[0,0]$
7: **end if**

Algoritmus 5a

Require: $n \geq 0, \mathbf{A}[2 \times 2]$
Ensure: $\mathbf{B} = \text{matpow}(\mathbf{A}, n)$

1: **if** $n > 1$ **then**
2: $\quad \mathbf{B} \Leftarrow \text{matpow}(\mathbf{A}, n/2)$
3: $\quad \mathbf{B} \Leftarrow \mathbf{BA}$
4: **end if**
5: **if** n je liché **then**
6: $\quad \mathbf{B} \Leftarrow \mathbf{A} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$
7: **end if**

Porovnání variant

Vlastnosti jednotlivých algoritmů

Algoritmus	Paměťové nároky	Časová složitost
1	$O(1)$	$O(\log N)$
2	$O(N)$	$O(F(N))$
3	$O(N)$	$O(N)$
4	$O(1)$	$O(N)$
5	$O(\log N)$	$O(\log N)$

Obsah přednášky

Asymptotická složitost
Velká $\mathcal{O}$ notace

Jakým způsobem se bude chování algoritmu měnit v závislosti na velikosti (počtu, objemu) vstupních dat?

Dva základní typy:

- **časová složitost** – vliv na dobu výpočtu
- **paměťová složitost** – nároky na operační paměť

Značíme:

- $\mathcal{O}(N)$ – lineární složitost,
- $\mathcal{O}(N^2)$ – kvadratická složitost,
- $\mathcal{O}(\log N)$ – logaritmická složitost.

Asymptotická složitost
Velká $\mathcal{O}$ notace

Vliv asymptotické časové složitosti

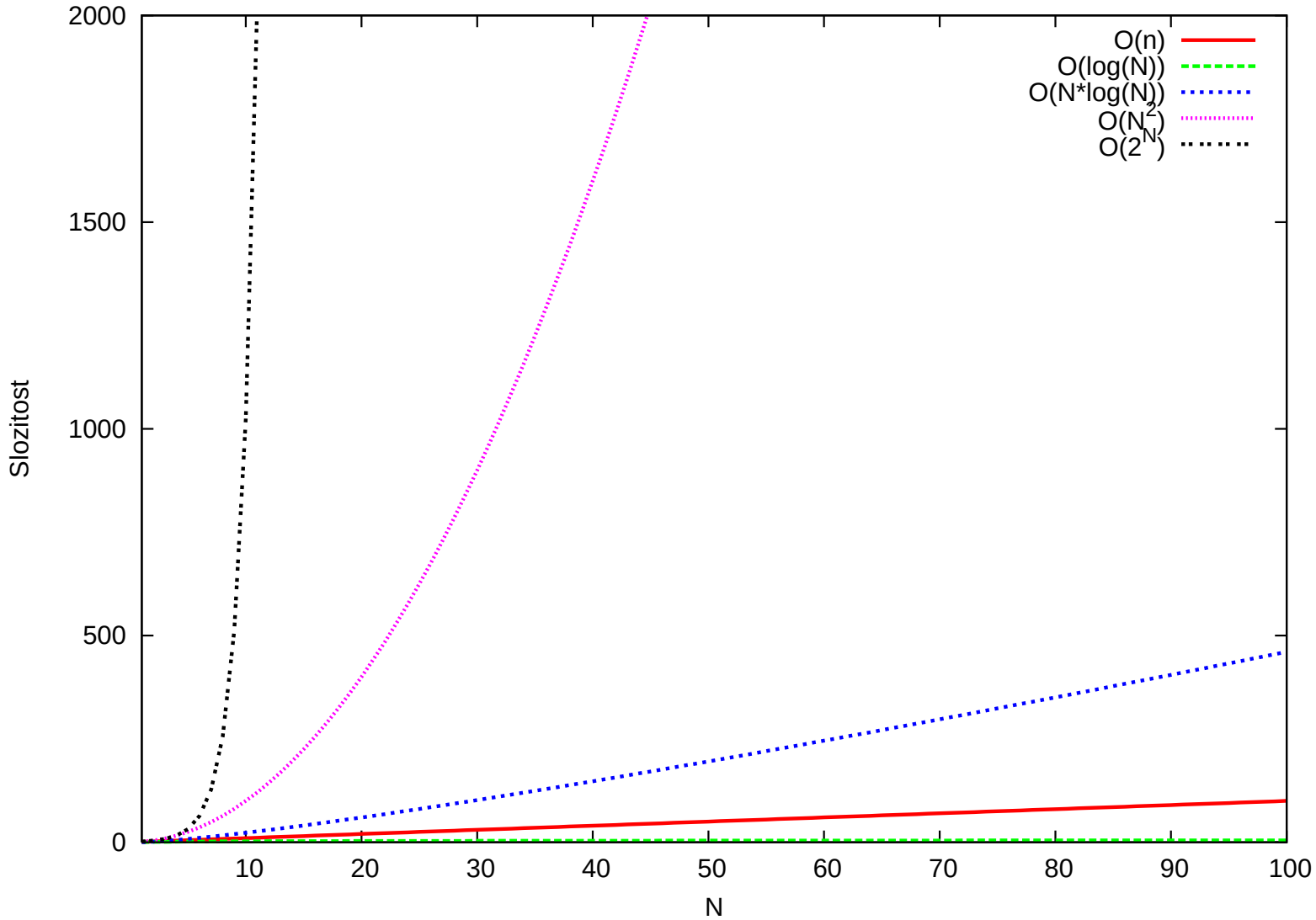
Pro $O(N^2)$ má zdvojnásobení objemu vstupních dat za následek čtyřnásobnou dobu vykonávání algoritmu.
Pro $O(\log N)$ může mít čtyřnásobný počet dat na vstupu za následek dvojnásobnou dobu vykonávání algoritmu.
Pro $O(1)$ je doba vykonávání algoritmu nezávislá na velikosti vstupu.

Vliv asymptotické paměťové složitosti

Pro $O(N)$ má zdvojnásobení velikosti vstupu za následek dvojnásob vysoké nároky na operační paměť.
Pro $O(2^N)$ čtyřnásobná velikost vstupu zosmínásobí paměťové nároky.

Asymptotická složitost

Obrázek



Obsah přednášky

Rozhodovací úlohy, třída $\mathcal{P}$

Velmi krátce z teorie výpočetní složitosti

Jako **rozhodovací úlohu** označujeme úlohu, jejímž řešením jsou výroky „ANO“ respektive „NE“.

Běžné úlohy v matematice lze snadno převést na rozhodovací úlohy.

Definice

Rozhodovací úloha L náleží do třídy $\mathcal{P}$, pokud existuje *deterministický* Turingův stroj, který tuto úlohu rozhodne v polynomiálním čase.

Příklady rozhodovacích úloh v třídě $\mathcal{P}$

Minimální kostra grafu – existuje kostra s ohodnocením menším, než c?

Nejkratší cesta v grafu – existuje cesta mezi dvěma uzly s ohodnocením menším, než c?

Lineární programování – existuje $\arg\max_{\boldsymbol{x}} \boldsymbol{w}^\top \boldsymbol{x} > c$ za daných omezujících podmínek?

Komprese dat (LZW) – přidá komprese řetězce s do slovníku slovo t?

Třída $\mathcal{NP}$

Nedeterministicky polynomiální úlohy

Definice

Rozhodovací úloha L náleží do třídy $\mathcal{NP}$, pokud existuje *nedeterministický* Turingův stroj, který tuto úlohu rozhodne v polynomiálním čase.

Nederministický Turingův stroj: vstupům může odpovídat více, než jedna jediná akce (sekvence $\Rightarrow$strom).

Platí $\mathcal{P} \subseteq \mathcal{NP}$.

Příklady rozhodovacích úloh v třídě $\mathcal{NP}$

Všechny úlohy třídy $\mathcal{P}$.

Izomorfismus grafu – lze dané dva grafy nakreslit stejně?

Faktorizace čísel – pro dané n a k, existuje $f : 1 < f < k, f|n$?

Všechny NP-úplné úlohy.

Třída NP-úplných úloh

Nejtěžší z třídy $\mathcal{NP}$

Třída NP-úplných problémů je třídou rozhodovacích úloh, pro něž platí následující definice:

Definice

Rozhodovací úloha L je NP-úplná, pokud náleží do třídy $\mathcal{NP}$ a zároveň jde o úlohu NP-těžkou

Co to znamená:

- Jakékoliv řešení L lze ověřit v polynomiálním čase
- Jakýkoliv problém z třídy NP lze převést na L transformací vstupů opět v polynomiálním čase

Tyto typy úloh umíme řešit pouze přibližně!

Příklady NP-úplných problémů

Problém batohu – lze zabalit batoh tak, aby jeho hmotnost nepřesáhla m a cena věcí byla alespoň c?

Problém obchodního cestujícího – existuje v grafu hamiltonovská kružnice o délce nejvýše c?

Obarvení grafu – lze uzly daného grafu obarvit nejvýše c barvami tak, aby sousedící uzly neměly stejnou barvu?

Problém čínského listonoše (pouze na smíšeném grafu) – existuje v grafu eulerovská kružnice o délce nejvýše c?